Pattern Recognition 131 (2022) 108846

Contents lists available at ScienceDirect

Pattern Recognition

journal homepage: www.elsevier.com/locate/patcog

A novel way to formalize stable graph cores by using matching-graphs☆

Mathias Fuchs[a,*], Kaspar Riesen[a,b]

[a] Institute of Computer Science, University of Bern, Bern 3012, Switzerland
[b] Institute for Information Systems, University of Appl. Sci. and Arts Northwestern Switzerland, Olten 4600, Switzerland

ARTICLE INFO

Article history:
Received 14 January 2022
Revised 18 May 2022
Accepted 9 June 2022
Available online 11 June 2022

Keywords:
Graph matching
Matching-graphs
Graph edit distance
Structural pattern recognition

ABSTRACT

The increasing amount of data available and the rate at which it is collected leads to rapid developments of systems for intelligent information processing and pattern recognition. Often the underlying data is inherently complex, making it difficult to represent it by linear, vectorial data structures. This is where graphs offer a versatile alternative for formal data representation. Actually, quite an amount of graph-based methods for pattern recognition has been proposed. A considerable part of these methods rely on graph matching. In the present paper, we propose a novel encoding of specific graph matching information. The basic idea is to formalize the stable cores of individual classes of graphs – discovered during intra-class matchings – by means of so called matching-graphs. We evaluate the benefit of these matching-graphs by researching two classification approaches that rely on this novel data structure. The first approach is a distance based classifier focusing on the matching-graphs during dissimilarity computation. For the second approach, we propose to use sets of matching-graphs to embed input graphs into a vector space. The basic idea is to produce hundreds of matching-graphs first, and then represent each graph g as a vector that shows the occurrence of, or the distance to, each matching-graph. In a thorough experimental evaluation on seven real world data sets we empirically confirm that our novel approaches are able to improve the classification accuracy of systems that rely on comparable information as well as state-of-the-art methods.

1. Introduction

Pattern recognition is a major field of research which aims at solving various problems like the recognition of facial expressions [1], the temporal sorting of images [2], or the enhancing of weakly lighted images [3], to name just a few examples. Pattern recognition can be roughly divided into two main approaches with respect to the formal data or pattern representation. *Statistical pattern recognition* relies on *feature vectors* for data representation, while *structural pattern recognition* employs *strings*, *trees*, or *graphs* for the same task. Since graphs are able to encode more information than merely an ordered and fixed-size list of real numbers, they offer a compelling alternative to vectorial approaches. This is particularly true in applications that involve complex data. Graphs are used in applications of highest diversity ranging from protein function/structure prediction [4], over signature verification [5], to the detection of Alzheimer's Disease [6], and many others [7,8]. The main drawback of graphs is, however, the computational complexity of basic operations, which in turn makes graph based algorithms for pattern recognition often slower than their statistical counterparts.

In the last four decades a huge number of procedures for *graph matching* have been proposed in the literature [7,8]. Graph matching is typically used for quantifying graph proximity. *Graph edit distance* [9], introduced about 40 years ago, is still recognized as one of the most flexible and robust graph matching models available. In contrast with many other distance measures (e.g. *graph kernels* [10] or *graph neural networks* [11]), graph edit distance offers more information than merely a dissimilarity score, viz. the information which subparts of the underlying graphs actually match with each other (known as *edit path*). To date we see no substantial research that exploits this particular knowledge as meta-information for reasoning about graphs and/or classifying graphs.

The present paper aims at bridging this gap. That is, we propose a specific encoding of matching information derived from graph edit distance in a novel data structure. In principle, the pro-

☆ Supported by Swiss National Science Foundation (SNSF) Project no. 200021_188496.
* Corresponding author
E-mail addresses: mathias.fuchs@inf.unibe.ch (M. Fuchs), kaspar.riesen@inf.unibe.ch, kaspar.riesen@fhnw.ch (K. Riesen).

https://doi.org/10.1016/j.patcog.2022.108846

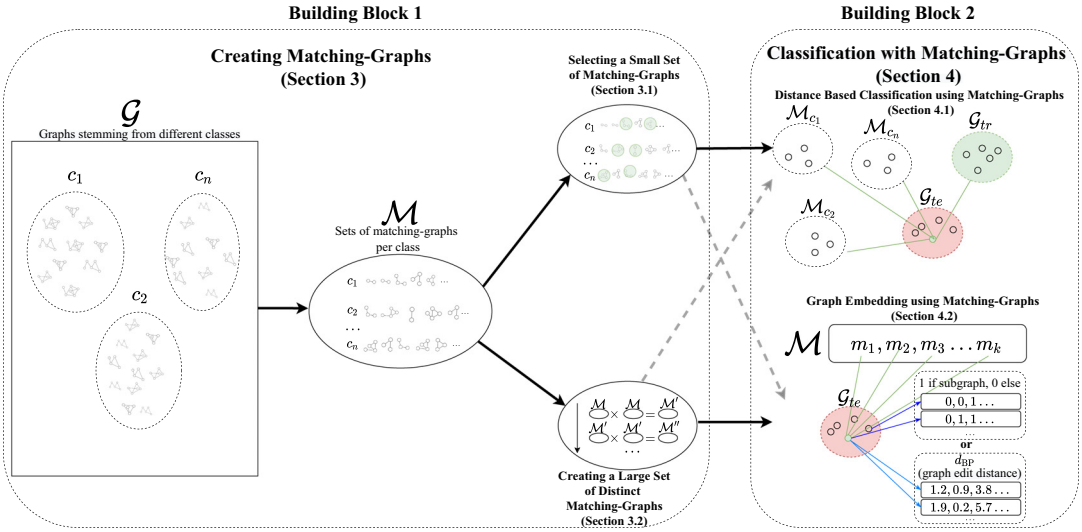

Fig. 1. The proposed framework consists of two major building blocks. The first is about creating matching-graphs (detailed in Section 3) and the second is about using the matching-graphs for classification (detailed in Section 4).

posed procedure first computes the graph edit distance for several pairs of labeled training graphs (stemming from the same class). Based on the matching found, a meta-graph is eventually created, which formalizes the corresponding parts of the two graphs. We denote this meta-graph as *matching-graph*. The rationale of these matching-graphs is to formalize the stable cores of specific classes of graphs. It is our hypothesis that the information captured in the resulting matching-graphs offers the potential to achieve both a better understanding of the regularities and stable parts of a given class and ultimately improve the matching quality of unknown patterns (by focusing on these stable cores of the graphs during the matching process, for instance). The overall goal of the present paper is to introduce matching-graphs, investigate their benefits, and ultimately verify our hypothesis.

Note that the present paper is based on three preliminary papers [12–14]. In [12] we describe the novel concept of matching-graphs for the first time and confirm the potential usefulness of these graphs in some initial experiments. In [13] we extend our initial idea by iteratively creating sets of matching-graphs on the basis of already existing matching-graphs. Finally, in Fuchs and Riesen [14] we propose to embed graphs into a vector space with the aid of matching-graphs. In the present work we combine the proposed methods in an overarching framework, give a more thorough and detailed description of the individual methods, and substantially extend both the method and the empirical evaluation. In particular, the present paper extends the previous papers as follows.

- We introduce a formal definition of the concept of a matching-graph, examine the background more thoroughly, and provide illustrative examples.
- Rather than only one method for selecting useful matching-graphs (as proposed in Fuchs and Riesen [12]), we now propose and evaluate two complementary selection algorithms.
- We simplify the iterative algorithm originally used in Fuchs and Riesen [13] so that it matches better to the requirement of creating very large sets of matching-graphs.
- In addition to the subgraph based embedding (as proposed in Fuchs and Riesen [14]) we now also define a distance based embedding using our matching-graphs.
- In contrast with the preliminary papers, we employ now two, rather than only one, classifiers in conjunction with our matching-graphs.
- We evaluate our algorithms on seven instead of only three data sets.
- We compare the different subsystems proposed in Fuchs and Riesen [12–14] for the first time with each other and show and discuss additional results. In particular, the present paper includes an ablation study which measures the value of the individual components of our framework, a comparison with state-of-the art algorithms from the field, and a run time analysis.

The remainder of the paper is organized as follows. Section 2 covers the basic definitions and concepts to make the paper self-contained. Moreover, it describes and summarizes the related work. In Section 3 we describe in detail the first of two major building blocks of our complete framework (illustrated in Fig. 1). In particular, we define how a set $\mathcal{M}$ of initial matching-graphs is built from arbitrary sets of graphs $\mathcal{G}$ and introduce two approaches to postprocess the initial matching-graphs. The first approach – described in Section 3.1 – condenses the set of matching-graphs by selecting a useful set of matching-graphs from the raw set. The second approach – described in Section 3.2 – is somehow complementary to the first approach since this method iteratively increases the initial set of matching-graphs. In Section 4, we describe the second building block of our novel framework, viz. the classification with matching-graphs. In particular, we introduce two conceptually different strategies to use the matching-graphs in order to solve classification problems. The first approach makes use of the matching-graphs in a distance based classifier (detailed in Section 4.1). The second classifier is built on a graph embedding that crucially relies on the matching-graphs (detailed in Section 4.2).

As shown in Fig. 1 we primarily use the matching-graphs selected with the method described in Section 3.1 for the distance based classifier. Likewise, the matching-graphs which are iteratively created by means of the method described in Section 3.2 are used for the graph embedding classifier. One could, however, also combine the method described in Section 3.1 with the classification method from Section 4.2 and, conversely, the method from Section 3.2 with the classification method from Section 4.1 (shown with dashed lines in Fig. 1). Although possible, we actually follow only those combinations that are connected with drawn lines (mainly for the sake of conciseness). The background and rationale for this decision follow in the corresponding subsections.

Eventually, in Section 5, we conduct a thorough experimental evaluation of our framework. That is, we empirically confirm that the matching-graphs are actually significant substructures of their classes and that our approach is able to improve the classification accuracy of various related systems. Finally, in Section 6, we conclude the paper and discuss potential ideas for future work.

2. Basic definitions and related work

2.1. Basic definitions

In this section, we provide the necessary definitions in order to make the paper self-contained.

Let L_V and L_E be finite or infinite label sets for nodes and edges, respectively. A *graph* g is a four-tuple $g = (V, E, \mu, \nu)$, where V is the finite set of nodes, $E \subseteq V \times V$ is the set of edges, $\mu : V \to L_V$ is the node labeling function, and $\nu : E \to L_E$ is the edge labeling function.

In some algorithms or applications it is necessary to formalize *empty "nodes"* and/or *empty "edges"* – both are denoted by symbol ε from now on. A *subgraph* g_1 of another graph g_2, denoted by $g_1 \subseteq g_2$, is defined in analogy to the subset relation in set theory. That is, a subgraph g_1 is obtained from a graph g_2 by removing some nodes and their incident (as well as possibly some additional) edges from g_2.

When graphs are used to represent different objects or patterns, a measure of distance or similarity is usually required [7]. We employ *graph edit distance* [9] as basic dissimilarity model throughout the paper. One of the main advantages of graph edit distance is its high degree of flexibility, which makes it applicable to virtually all types of graphs.

Given two graphs g_1 and g_2, the basic idea of graph edit distance is to transform g_1 into g_2 using some *edit operations* for both nodes and edges (such as *insertions, deletions*, and *substitutions*). We denote the substitution of two nodes $u \in V_1$ and $v \in V_2$ by $(u \to v)$, the deletion of node $u \in V_1$ by $(u \to \varepsilon)$, and the insertion of node $v \in V_2$ by $(\varepsilon \to v)$. For edge edit operations we use a similar notation.

A set $\{e_1, \ldots, e_k\}$ of k edit operations e_i that transform a source graph g_1 completely into a target graph g_2 is called an *edit path* $\lambda(g_1, g_2)$ between g_1 and g_2. Let $\Upsilon(g_1, g_2)$ denote the set of all edit paths transforming g_1 into g_2 while c denotes the cost function measuring the strength $c(e_i)$ of edit operation e_i. The graph edit distance can now be defined as follows.

Let $g_1 = (V_1, E_1, \mu_1, \nu_1)$ be the source and $g_2 = (V_2, E_2, \mu_2, \nu_2)$ the target graph. The *graph edit distance* between g_1 and g_2 is defined by

$$d_{\lambda_{\min}}(g_1, g_2) = \min_{\lambda \in \Upsilon(g_1, g_2)} \sum_{e_i \in \lambda} c(e_i) \tag{1}$$

Optimal algorithms for computing the edit distance of two graphs are typically based on combinatorial search procedures. A major drawback of those procedures is their computational complexity, which is exponential in the number of nodes. To render graph edit distance computation less demanding, approximation algorithms can be employed (e.g. Riesen and Bunke [15], Fischer et al. [16], Bougleux et al. [17]). These algorithms offer cubic or quadratic time complexity with respect to the number of nodes of the involved graphs.

Actually, any computation method for graph edit distance can be employed as basis in our framework, as long as it provides us with a valid edit path between two graphs. That is, the actual algorithm for graph matching does not crucially impact the rest of the proposed method. We decide to use the graph edit distance approximation termed *BP* [15], since it is a widely used (and somehow a standard) algorithm for the graph edit distance approximation. The graph edit distance between g_1 and g_2 computed by algorithm BP is termed $d_{\mathrm{BP}}(g_1, g_2)$ from now on.

2.2. Related work and broader perspective

The concept of matching-graphs actually requires a graph matching procedure – the task of identifying similar substructures in two graphs. We make use of graph edit distance [9] for this basic task (outlined above). Over the years, however, several other dissimilarity measures for graphs have been proposed [7,8]. They range from Spectral Methods [18], over Graduated Assignment Algorithms [19], to Expectation Maximization Algorithms and Continuous Optimization Algorithms [20].

Some of the most prominent graph matching algorithms are *graph kernels*. A seminal contribution in the field are kernels that are based on the analysis of walks or paths in graphs. These kernels measure the similarity of two graphs by the number of equal (or at least similar) walks or paths in the underlying graphs [10]. In [4] a second class of kernels for graphs with discrete labels is introduced. This class of kernels is based on the 1-dimensional Weisfeiler–Lehman, or color refinement, algorithm. Since this contribution, several extensions and adaptations of this idea have been proposed [21]

A further prominent class of graph kernels is based on the work on convolution kernels, which provide a general framework for dealing with complex objects that consist of simpler parts. In particular, convolution kernels infer the similarity of two objects from the similarity of their parts (e.g., nodes, subgraphs, or trees [10]).

Graph embedding approaches can actually also be interpreted as graph kernels. In [22], for instance, a graph g is represented by a vector that counts the number of times certain subgraphs occur in g, while the *Subgraph Matching Kernel* [23] and *Graphlet Kernel* [24] both count the number of matchings between subgraphs of fixed sizes in two graphs. In [25], a graph is represented based on its dissimilarities to certain prototypes.

Besides the strong dependency of our novel matching-graphs on a specific graph matching procedure, we also observe a certain similarity of our novel concept with the idea of *Frequent Subgraph Mining (FSM)* [26]. FSM focuses on the identification of frequent subgraphs within a set of graphs. In particular, in FSM, one aims at extracting all subgraphs from a given set of graphs that occur more often than a specified threshold. We observe two main categories in FSM, viz. *Apriori-based approaches* [27] and *Pattern-growth approaches* [28]. The apriori-based methods start with frequent nodes and proceed to grow subgraphs by using a *Breadth First Search* strategy. That is, before they continue to find graphs of size $k+1$ these approaches first search for all frequent graphs of size k. Pattern-growth approaches, on the other hand, work by using a *Depth First Search* strategy, where one graph is extended until all frequent supergraphs of this graph are found.

3. Creating matching-graphs

The general idea of *matching-graphs* – originally proposed in Fuchs and Riesen [12] – is to extract information on the matching of pairs of graphs in a new data structure that in turn encodes the corresponding parts of the two graphs.

Formally, we assume k sets of training graphs $G_{\omega_1}, \ldots, G_{\omega_k}$ stemming from k different classes $\omega_1, \ldots, \omega_k$. We formalize the information on the matching of two graphs $g_i = (V_i, E_i)$, $g_j = (V_j, E_j)$ – stemming from the same class ω_l – in a graph denoted by $m_{g_i \times g_j}$.

Basically, a matching-graph $m_{g_i \times g_j}$ should represent both nodes and edges of g_i and g_j that have been matched under the usage of some particular graph matching algorithm. In our scenario, matching-graphs $m_{g_i \times g_j}$ are created according to the following procedure.

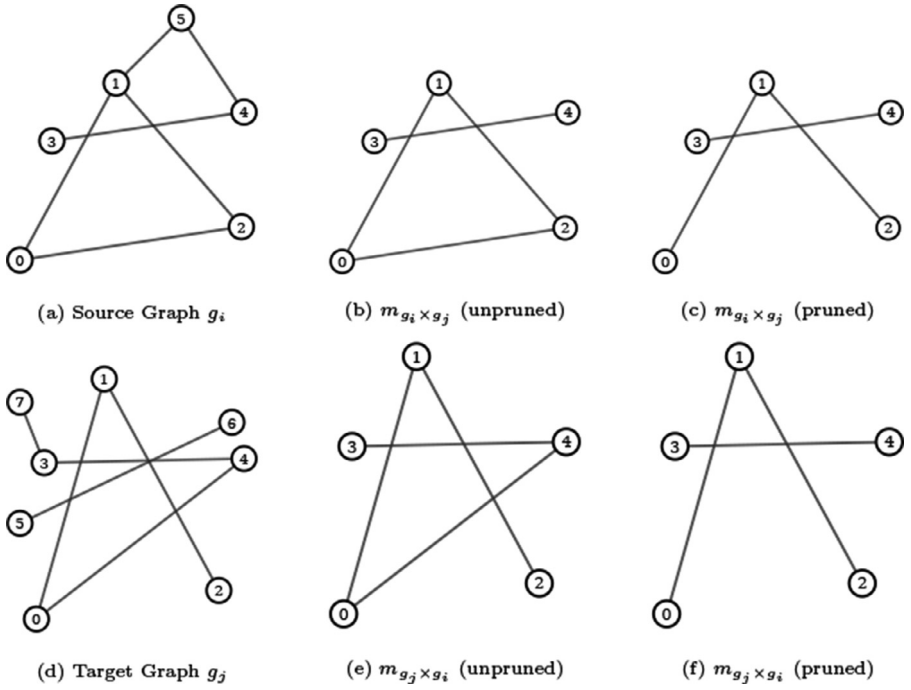

Fig. 2. Matching-Graphs with unpruned and pruned edges derived from source graph g_i and target graph g_j.

For all pairs of graphs stemming from the same class ω_l, the graph edit distance is computed by means of algorithm BP [15]. Hence, we obtain a (sub-optimal) edit path $\lambda(g_i, g_j)$ for each pair of graphs g_i, g_j. For each edit path $\lambda(g_i, g_j)$, two matching-graphs $m_{g_i \times g_j}$ and $m_{g_j \times g_i}$ are built (for the source and the target graph g_i and g_j, respectively). To this end, all nodes of g_i and g_j that are actually substituted in edit path $\lambda(g_i, g_j)$ are added to $m_{g_i \times g_j}$ and $m_{g_j \times g_i}$, respectively. Vice versa, all nodes that are deleted in g_i or inserted in g_j are neither considered in the two matching-graphs.

In preliminary experiments we observe that isolated nodes might occur in the resulting matching-graphs. Although many graph matching algorithms can actually handle isolated nodes, we still remove them from our matching-graphs. The rationale for this heuristic is that we aim at building small and robust cores of the graphs with nodes that are actually connected to at least one other node in the formal substructure.

The question remains how to handle the edges of the involved graphs g_i, g_j in the resulting matching-graphs $m_{g_i \times g_j}$ and $m_{g_j \times g_i}$. Clearly, if a node is not included in the matching-graph (since it was either deleted or inserted in the underlying edit path), the incident edges of this node will not be included in the resulting matching-graph as well. Edges that connect two substituted nodes, however, can be included in the matching-graphs. We propose two different strategies for edge handling.

1. *No Pruning*: If two nodes $u_1, u_2 \in V_i$ of a source graph g_i are substituted with nodes $v_1, v_2 \in V_j$ in a target graph g_j and there is an edge $(u_1, u_2) \in E_i$ available, (u_1, u_2) is actually included in the matching-graph $m_{g_i \times g_j}$ regardless whether or not edge (v_1, v_2) is available in E_j. Hence, in this case *no pruning* is applied to the edges.
2. *Pruning*: We assume the same scenario as above. However, edge (u_1, u_2) is included in the matching-graph $m_{g_i \times g_j}$ if, and only if, there is an edge (v_1, v_2) available in E_j. Hence, in cases where no corresponding edge can be found in the other graph, the edge is actually *pruned*.

Formally, a matching-graph $m_{g_i \times g_j} = (V_{g_i \times g_j}, E_{g_i \times g_j})$ is defined as

- $V_{g_i \times g_j} = \{v \in V_i : (v \to u) \in \lambda(g_i, g_j) \text{ and } u \in V_j\}$
- Unpruned: $E_{g_i \times g_j} = \{E_i \cap (V_{g_i \times g_j} \times V_{g_i \times g_j})\}$
- Pruned: $E_{g_i \times g_j} = \{E_i \cap E_j \cap (V_{g_i \times g_j} \times V_{g_i \times g_j})\}$

For the matching-graph $m_{g_j \times g_i}$ the definition is similar to $m_{g_i \times g_j}$, but the indices i and j have to be exchanged. From a broader perspective, the novel matching-graphs can be interpreted as a generalization of the concept of a *common subgraph* [29]. In its original definition, a common subgraph of two graphs consists of nodes which occur identically in the both graphs. In our novel data structure, a node is incorporated whenever the corresponding node is actually substituted with another node w.r.t. the found edit path.

In Fig. 2 an illustration of the procedure is given for two graphs of the Letter graph data set (graphs from this data set represent artificially distorted letter line drawings, and are often used for illustration purposes [30]). For this example the matching between the source and target graph results in the edit path $\lambda(g_i, g_j) = \{0 \to 0, 1 \to 1, 2 \to 2, 3 \to 3, 4 \to 4, 5 \to \varepsilon, \varepsilon \to 5, \varepsilon \to 6, \varepsilon \to 7\}$. According to this edit path, the two matching-graphs that are generated without pruning are shown in Fig. 2 (b) and (e). By applying edge pruning, we observe that edges that have no counterpart in the other graph are not included in the resulting matching-graph (like, for instance, the edges $(0, 2)$ or $(0, 4)$ in the source and target graphs, respectively). Regardless the strategy actually applied, we observe a strong denoising effect on the input graphs in this illustrative example.

The actual definition of the cost function, and in particular the cost for insertions and deletions of nodes, has a crucial impact on the resulting matching-graphs. The higher the cost for node deletions/insertions is defined, the more nodes of both graphs are substituted with each other, which in turn leads to larger matching-graphs in general. This effect is illustrated in Fig. 3. We show different matching-graphs derived from two source graphs (representing the letters A and E). We use different cost values for node deletions/insertions. By decrementing the deletion/insertion cost we gradually obtain smaller matching-graphs (with fewer and fewer nodes in general).

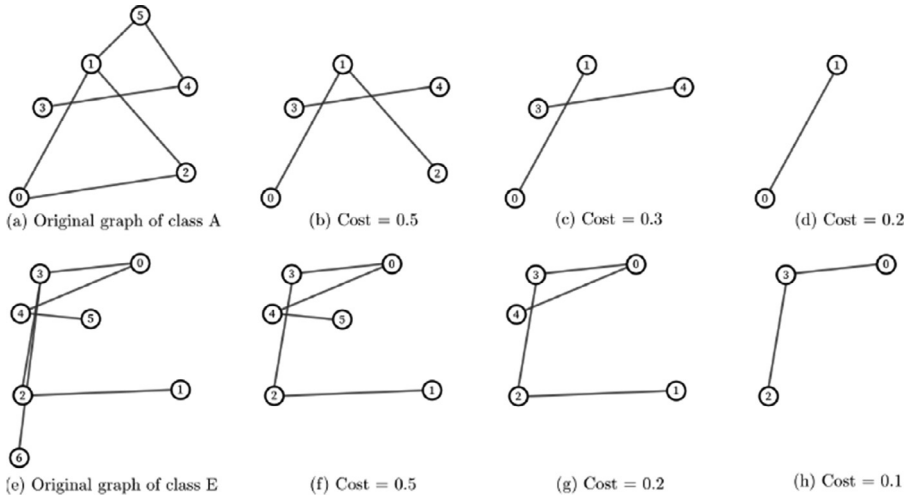

Fig. 3. The smaller the cost for both node deletion and insertion is defined, the smaller is the resulting matching-graph in general.

3.1. Selecting a small set of matching-graphs

Let us assume we have a set of training graphs G_{ω_l} available. Furthermore, we assume that G_{ω_l} contains n graphs representing class ω_l. If we create all possible matching-graphs for all possible combinations of graph pairs (g_i, g_j) stemming from $G_{\omega_l} \times G_{\omega_l}$, we end up with a set of matching-graphs $\mathcal{M}_{\omega_l}$ of size $n(n-1)$. Depending on both the actual size of G_{ω_l} and the specific requirements for $\mathcal{M}_{\omega_l}$ this quantity might be too large[1]. In order to reduce $\mathcal{M}_{\omega_l}$ to a reasonable size, various graph selection methods can be used [25]. We propose to reduce $\mathcal{M}_{\omega_l}$ with the aid of the *set median graph* [31], which is defined as

$$median(\mathcal{S}) = \underset{g_1 \in \mathcal{S}}{\operatorname{argmin}} \sum_{g_2 \in \mathcal{S}} d(g_1, g_2)$$

where $\mathcal{S}$ is an arbitrary set of graphs. The set median graph is the graph of $\mathcal{S}$ whose sum of distances to all other graphs in $\mathcal{S}$ is minimal.

Based on the set median graph we propose two ways to select matching-graphs.[2] Both approaches take as input an initial set of matching-graphs $\mathcal{M}_{\omega_l}$ and a user defined parameter t which corresponds to the number of matching-graphs desired. The first algorithm, iteratively selects (and eventually removes) the set median graph from the set of all available matching-graphs $\mathcal{M}_{\omega_l}$ until the required number t of matching-graphs is selected (see Algorithm 1). That is, we select in total t matching-graphs that are situated in, or near, the center of the complete set of matching-graphs.

Algorithm 1: Center-selection $(\mathcal{M}_{\omega_l}, t)$.

1: Initialize $\bar{\mathcal{M}}_{\omega_l}$ to the empty set {}
2: **while** $|\bar{\mathcal{M}}_{\omega_l}| < t$ **do**
3: $m = median(\mathcal{M}_{\omega_l})$
4: $\bar{\mathcal{M}}_{\omega_l} = \bar{\mathcal{M}}_{\omega_l} \cup \{m\}$
5: $\mathcal{M}_{\omega_l} = \mathcal{M}_{\omega_l} \setminus \{m\}$
6: **end while**
7: **return** $\bar{\mathcal{M}}_{\omega_l}$

Second, we propose a spanning based approach. We also start by selecting the set median graph. Each additional matching-graph selected is the graph the furthest away from the already selected matching-graphs. We repeat this procedure until the required number t of matching-graphs is selected (see Algorithm 2).

Algorithm 2: Spanning-selection $(\mathcal{M}_{\omega_l}, t)$.

1: Initialize $\bar{\mathcal{M}}_{\omega_l}$ to the empty set {}
2: $m = median(\mathcal{M}_{\omega_l})$
3: $\bar{\mathcal{M}}_{\omega_l} = \bar{\mathcal{M}}_{\omega_l} \cup \{m\}$
4: $\mathcal{M}_{\omega_l} = \mathcal{M}_{\omega_l} \setminus \{m\}$
5: **while** $|\bar{\mathcal{M}}_{\omega_l}| < t$ **do**
6: $m = \underset{g \in \mathcal{M}_{\omega_l}}{\operatorname{argmax}} \min_{m \in \bar{\mathcal{M}}_{\omega_l}} d(g, m)$
7: $\bar{\mathcal{M}}_{\omega_l} = \bar{\mathcal{M}}_{\omega_l} \cup \{m\}$
8: $\mathcal{M}_{\omega_l} = \mathcal{M}_{\omega_l} \setminus \{m\}$
9: **end while**
10: **return** $\bar{\mathcal{M}}_{\omega_l}$

3.2. Creating a large set of distinct matching-graphs

The overall aim of the two methods described in the previous subsection is to reduce the set of matching-graphs to a reasonable size. Depending on the actual application and requirements it might be beneficial to have a large set of matching-graphs that are distinct from each other. For this purpose, we propose an iterative algorithm to produce matching-graphs out of existing matching-graphs.[3] Algorithm 3 takes as input k sets of graphs $G_{\omega_1}, \ldots, G_{\omega_k}$ with graphs from different classes $\omega_1, \ldots, \omega_k$, as well as the number of matching-graphs n that will be kept from one iteration to another (n is a user defined parameter).

The algorithm iterates over all k sets (classes) of graphs from $G \in \mathcal{G}$ (main loop of Algorithm 3, from line 2 to line 14). For each set of graphs G and for all possible pairs of graphs g_i, g_j stemming from the current set G, the initial set of matching-graphs M is produced (line 3 to 6).[4] Note that a matching-graph is only added to

[1] Note that the proposed framework can instantly produce matching-graphs for any pair of graphs, and is thus not reliant on a specific set or subset of graphs. This, in turn, makes our system quite fast and flexible (since we only need to consider pairs of graphs to create a new graph). Moreover, since it is possible to specify in advance how many graphs of the training set are actually used to create the matching-graphs, scalability is not a major problem in practical applications.

[2] In contrast to Fuchs and Riesen [12], where only one approach is proposed and evaluated.

[3] The method described in the present section is similar to the algorithm proposed in a preliminary paper [13]. In contrast with [13], however, we propose a simplification of the algorithm so that we get more and also distinct graphs.

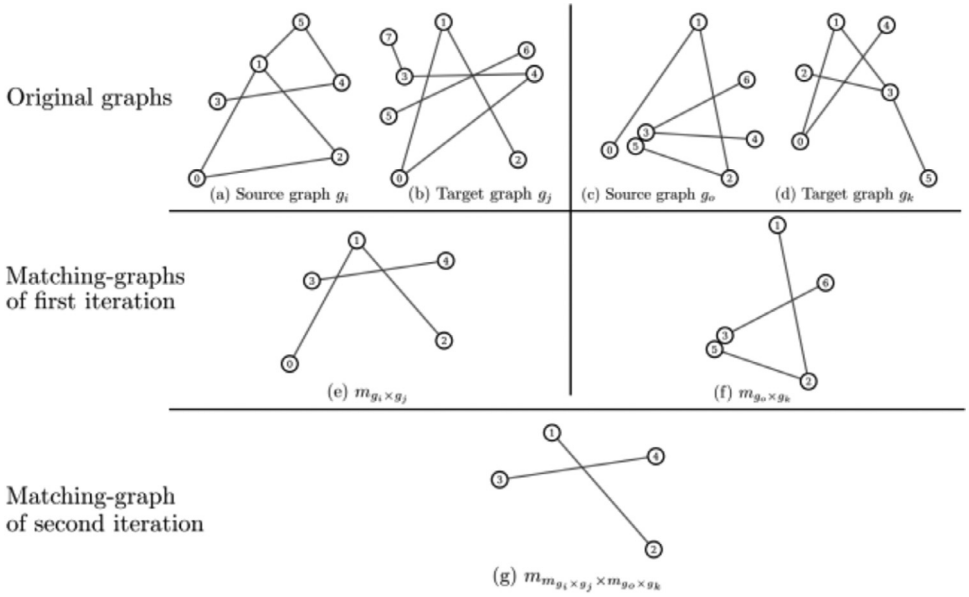

Fig. 4. Illustration of the procedure of creating matching-graphs over multiple iterations.

Algorithm 3: Iterative matching-graph creation.

input : sets of graphs from k different classes $\mathcal{G} = \{G_{\omega_1}, \ldots, G_{\omega_k}\}$, the maximum number n of matching-graphs to keep in each iteration
output: sets of matching-graphs for each of the k different classes $\mathcal{M} = \{M_{\omega_1}, \ldots, M_{\omega_k}\}$

1 Initialize $\mathcal{M}$ as the empty set: $\mathcal{M} = \{\}$
2 **foreach** *set of graphs* $G \in \mathcal{G}$ **do**
3 Initialize M as the empty set: $M = \{\}$
4 **foreach** *pair of graphs* $g_i, g_j \in G \times G$ *with* $j > i$ **do**
5 $M = M \cup \{m_{g_j \times g_i}, m_{g_i \times g_j}\}$
6 **end**
7 **do**
8 $M' =$ a subset of n randomly selected graphs of M
9 **foreach** *pair of graphs* $m_i, m_j \in M' \times M'$ *with* $j > i$ **do**
10 $M = M \cup \{m_{m_j \times m_i}, m_{m_i \times m_j}\}$
11 **end**
12 **while** M *has changed in the last iteration*
13 $\mathcal{M} = \mathcal{M} \cup M$
14 **end**

M if it does not already exist in M, meaning that we do not allow to have duplicates in M. Eventually, we aim at iteratively building matching-graphs out of pairs of existing matching-graphs. The motivation for this procedure is to further reduce the size of the matching-graphs and to find small core-structures that are often available in the corresponding graphs. Due to computational limitations, we randomly select a subset of size n from the current matching-graphs M (line 8). Based on this selection, the next generation of matching-graphs is built. This is actually carried out in the second for-loop on lines 9 to 11 where for all pairs of graphs in M' two novel matching-graphs are created and added to M. This process is repeated until no more changes occur in set M. Finally, set $\mathcal{M}$ is compiled as the union of all matching-graphs individually produced for all available classes.

In Fig. 4 we provide an illustrative example of our iterative procedure on four graphs from the Letter data set [30]. Subfigures (a) to (d) show the original graphs. Subfigure (e) shows the resulting matching-graph $m_{g_i \times g_j}$ of graph g_i and g_j, whereas Subfigure (f) shows the matching-graph $m_{g_o \times g_k}$ resulting from graphs g_o and g_k. Finally, in Subfigure (g) we show the matching-graph resulting from the two matching-graphs of the first iteration.[5] This example illustrates that the size of the matching-graphs declines from one iteration to another in general.

4. Classification with matching-graphs

We propose two approaches for using the matching-graphs in a classification scenario. The first idea is to enhance the accuracy of graph edit distance by explicitly focusing on matching-graphs (detailed in Section 4.1). The second idea is to use the resulting matching-graphs for graph embedding (detailed in Section 4.2).

4.1. Distance based classification using matching-graphs

In our first approach we use the matching-graphs in a distance based classification scenario. Let us assume we aim at computing the distance between a given test graph g and a training graph $g_i \in G_{\omega_l}$. We define a novel distance measure $d_{\mathrm{M}}(\cdot,\cdot)$ that combines the following two distances with each other.

1. The approximated graph edit distance information $d_{\mathrm{BP}}(g, g_i)$ between the test graph g and the original training graph $g_i \in G_{\omega_l}$.
2. A statistical score S on the basis of *all* distances between the test graph g and all matching-graphs $m \in \mathcal{M}_{\omega_l}$ stemming from the set of matching-graphs of class ω_l (the actual class of the corresponding training graph g_i).

Formally, we define the distance d_{M} as a weighted sum of the original edit distance and the information obtained by means of the meta-matching. That is,

$$d_{\mathrm{M}}(g, g_i) = \alpha \cdot d_{\mathrm{BP}}(g, g_i) + (1 - \alpha) \cdot S(\{d_{\mathrm{BP}}(g, m) : m \in \mathcal{M}_{\omega_l}\})$$

where $\alpha \in [0, 1]$ is a weighting parameter to trade off between the two dissimilarity scores and function S denotes a descriptive sta-

[4] We take into account the first matching-graph $m_{g_i \times g_j}$ only. Moreover, due to computational reasons we stick with the pruned version of the matching-graphs.

[5] The matching-graph in Fig. 4(g) appears very small and generic and not specific to the class. Note, however, that the specificity heavily depends on the node labels. Also keep in mind, that this is only an illustrative example with rather small graphs in order to give an intuition.

tistical value computed on the set of distances between the original graph g and the matching-graphs $m \in \mathcal{M}_{\omega_l}$ (we propose to use the minimum, the maximum, or the average function for S). Clearly, with $\alpha = 1$ we obtain the standard graph edit distance, while $\alpha = 0$ leads to a distance that relies on the matching-graphs only.

The set of matching-graphs $\mathcal{M}_{\omega_l}$ actually used for building d_{M}, is created and reduced according to the process described in Section 3.1. It would also be possible to use the large sets of matching-graphs iteratively created by means of the method described in Section 3.2 (it actually turns out that this produces similar results as the proposed combination). However, this specific setup is computationally much more demanding – due to the very large set of matching-graphs – and is therefore not a viable alternative which is not pursued.

We employ this novel distance model in two classifiers. First, we feed d_{M} into a k-NN classifier denoted by k-NN(d_{M}). Second, we use the novel distance as basic similarity kernel $\kappa(g_i, g_j) = -d_{\mathrm{M}}(g_i, g_j)$ in conjunction with a Support Vector Machine (denoted as SVM($-d_{\mathrm{M}}$)).

4.2. Graph embedding using matching-graphs

The general idea of the second classification approach is to embed a given graph into a vector space by means of the matching-graphs. Let g be an arbitrary graph stemming from a given set of graphs. Using a large set $\mathcal{M} = \{m_1, \ldots, m_N\}$ of N matching-graphs, created according to the method described in Section 3.2. One could also employ the matching-graph selection described in Section 3.1 for this purpose. The rationale of this selection is, however, to reduce an existing set. For embedding, on the other hand, we are more interested in generating large sets of distinct graphs. Therefore, this particular combination seems a bit counter-intuitive. Moreover, we observe that both combinations – given that the used sets are actually large enough – achieve quite similar results. Hence, we only follow one of the two possible combinations.

We embed g in two different ways. Once using subgraph isomorphism (as originally proposed in Fuchs and Riesen [14]) and once using the graph edit distance.

The first embedding is defined by

$$\varphi_{sub}(g) = (sub(m_1, g), \ldots, sub(m_N, g)),$$

where $sub(m_i, g) = 1$, if $m_i \subseteq g$, and 0 else.

That is, for this embedding we employ subgraph isomorphism that provides us with a binary similarity measure which is 1 or 0 for subgraph-isomorphic and non-subgraph-isomorphic graphs, respectively. When considering if a given matching-graph m_i is a subgraph of a graph g, it is necessary to decide whether or not two nodes are equal with respect to their labels. For nodes with categorical labels this task can be solved in a straightforward manner. When a node, however, contains continuous labels, this decision process is more subtle. In this particular case one could, for instance, employ a distance measure for the node labels and eventually define a threshold to decide whether or not two nodes are similar enough to be considered as equal.

There are various algorithms available that can be applied to solve the subgraph isomorphism problem [7,8]. In the present paper we employ the VF2 algorithm [32] which makes use of efficient heuristics to speed up the search process.

The second embedding is defined by

$$\varphi_{ged}(g) = (d_{\mathrm{BP}}(g, m_1), \ldots, d_{\mathrm{BP}}(g, m_N))$$

In other words we compute the graph edit distance (in our case using the suboptimal algorithm BP [15]), between the graph g to be embedded and all matching-graphs in $\mathcal{M}$ and then represent g as a vector of the resulting distances.

Obviously, both graph embeddings produce vectors with a dimension that is equal to the number of matching-graphs actually available. As the iterative method described in Section 3.2 might generate thousands of matching-graphs, the dimension of the resulting feature vectors might be very large. In cases where the high dimensionality of the data is a problem, one can apply an arbitrary feature selection method to the resulting graph embeddings.

These specific graph embeddings are similar in spirit to the frequent substructure approaches [22], the subgraph matching kernel [23], the graphlet kernel [24], as well as dissimilarity based embeddings [25]. The main difference of our approach to those methods lies in the creation of the subgraphs (or prototypes in case of [25]). We employ graph edit distance to create our novel data structure of matching-graphs. These matching-graphs offer a natural way of defining significant and large sets of subgraphs that can readily be used for vector space embeddings.

Likewise to the novel distance d_{M}, also for the graph embedding we employ two different classifiers. First, we classify the resulting vectors using a k-NN in conjunction with a vector similarity measure s. For the subgraph based embedding $\varphi_{sub}(g)$ we use binary similarity measures *Dice, Yule, Tanimoto (Rogers), Jaccard coefficient*, as well as *Kulczynski-1* and *2*. For the distance based embedding $\varphi_{ged}(g)$ we use the *Euclidean, Cosine* and *Minkowski* dissimilarity. Second, we employ an SVM that operates on the embedding vectors (using standard kernel functions k for feature vectors such as the *Radial Basis Function (RBF)*, the *Sigmoid kernel*, and the *Linear kernel*). We denote these approaches as k-NN($s_{\varphi(g)}$) and SVM($\kappa_{\varphi(g)}$), respectively.

5. Experimental evaluation

5.1. Experimental setup

The main question to be answered in our empirical evaluation is whether the proposed matching-graphs can be used to improve the classification accuracies of existing graph matching procedures (that rely on the same graphs and graph edit distance information as our novel procedure). Hence, we compare our novel method with two reference classifiers that are often used in conjunction with graph edit distance.

The first reference system is a k-nearest-neighbor classifier (k-NN) that directly operates on the distances d_{BP}, denoted as k-NN(d_{BP}) from now on. The second reference system is a Support Vector Machine, denoted as SVM($-d_{\mathrm{BP}}$), that operates on the similarity kernel $\kappa(g_i, g_j) = -d_{\mathrm{BP}}(g_i, g_j)$.

5.1.1. Data sets

The novel approaches for graph classification using matching-graphs are evaluated on seven data sets.

- **AIDS, Mutagenicity, NCI1, PTC(MR) and COX-2**: We use five data sets that represent chemical compounds from different applications.The AIDS data set stems from the IAM graph repository [30] and was initially gathered by the National Cancer Institute (NCI). The compounds that are able to perfectly protect the human cells from HIV are labeled as *Confirmed Active* and those that are not are labeled *Confirmed Inactive*. The *Mutagenicity* data set [30] is also split into two classes, one containing mutagenic compounds, and the other non-mutagenic compounds. The other data sets from this category stem from Morris et al. [33]. The NCI1 data set originates from anti-cancer screens and is split into molecules that contain activity for growth inhibition of non-small cell lung cancer (*active*) and these that do not (*inactive*). The fourth data set *PTC(MR)* consists of compounds that are potentially carcinogenic. The *COX-2*

Table 1
The total number of graphs for each data set as well as the corresponding number of graphs in the training, validation, and test sets.

Data set	Total	Training	Validation	Test
AIDS	2000	250	250	1500
Mutagenicity	4337	1500	500	2337
NCI1	4110	2465	822	823
IMDB	1000	600	200	200
COX-2	466	280	93	93
PTC (MR)	344	206	68	70
Letter	2250	750	750	750

data set contains cyclooxygenase-2 (COX-2) inhibitors with or without in-vitro activities against human recombinant enzymes. The nodes of the chemical compound data sets always represent the atoms labeled with their chemical symbol. The edges of the graphs of the PTC(MR) data set additionally contain information about the chemical bonds between the atoms, while the edges of all other sets remain unlabeled.

- **IMDB**: The IMDB data set stems from Morris et al. [33] and is a movie collaboration data set with the actor/actresses and genre information of a movie. The nodes represent the actors/actresses and the edges state whether they appear in the same movie. A single graph always represents one movie stemming from a genre, which is either action or romance (the class to be predicted).
- **Letter**: This data set contains graphs that represent artificially distorted letter line drawings of 15 different letters that consist of straight lines [30]. The nodes represent the end points of individual lines, and the edges represent the line drawing itself. The nodes are labeled with their corresponding x-and y-coordinates. This data set is often used for preliminary evaluations and, moreover, employed to provide illustrative examples of novel techniques (as we also did in the previous sections).

5.2. Validation of metaparameters

For the experimental evaluation each data set is split into three predefined random disjoint sets for training, validation, and testing. Details about the size of the individual splits can be found in Table 1. The matching-graphs are created on the training set only, whereas the optimization of the metaparameters is performed with the help of the validation set. The optimal parameters obtained with the usage of the validation set are then applied on the test set (without any further modifications).

For algorithm BP, that approximates the graph edit distance, the cost for node and edge deletions, as well as a weighting parameter $\beta \in [0, 1]$ that is used to trade-off the relative importance of node and edge edit costs are often optimized [15,25]. However, for the sake of simplicity we employ unit cost of 1.0 for deletions and insertions of both nodes and edges and optimize the weighting parameter β only (on all data sets). For data sets where the underlying graphs contain label alphabets L_ν with categorical attributes, the cost of non-identical substitutions is set to the cost of one insertion plus the cost of one deletion (which amounts to 2). For data sets with continuous node labels, we employ the Euclidean distance as a cost for substituting the two nodes. For the creation of the matching-graphs – actually also dependant on the cost model – the same cost parameters are employed.

For both classification algorithms k-NN(d_{M}) and SVM($-d_{\mathrm{M}}$) we optimize the weighting parameter α (used in d_{M}), the type of matching-graphs (pruned vs. unpruned), the matching-graph selection method (*center* vs. *spanning*), the number t of selected matching-graphs per class, and function S, that determines whether the minimum, the maximum, or the average is used to condense the set of distances $\{d_{\mathrm{BP}}(g, m) : m \in \mathcal{M}_{\omega_l}\}$. In addition, for the k-NN classifier we optimize the number of neighbors k considered, while for the SVM classifier parameter C is optimized to trade off between the size of the margin and the number of misclassified training examples. The discussion of the validation results and the actual parameter values can be found in Appendix A.1.

Both classification algorithms k-NN($s_{\varphi(g)}$) and SVM($\kappa_{\varphi(g)}$) rely on graph embeddings $\varphi(g)$ that are computed by means of large sets of matching-graphs. Remember that these sets are created in an iterative manner. We set the number of matching-graphs considered for the next iteration to $n = 200$ on all data sets. The stop criterion of the iterative process checks whether or not the last iteration resulted in a change of the currently considered set of matching-graphs and the dimension of the resulting feature vectors turns out to be very large. Thus, we apply a recursive feature elimination process [34] to the resulting graph embeddings. After the feature selection process, we end up with about 5 to 13% of the available matching-graphs for all data sets. The complete analysis on the feature selection process can be found in A.2.

For both approaches k-NN($s_{\varphi(g)}$) and SVM($\kappa_{\varphi(g)}$) we optimize the type of the embedding ($\varphi_{sub}(g)$ vs. $\varphi_{ged}(g)$) as well as the weighting parameter $\beta \in [0, 1]$ that is used to trade-off the relative importance of node and edge edit costs. For the k-NN we further optimize the similarity measure s as well as the number k of neighbors that are considered. For the SVM we optimize the kernel function and parameter $C \in [0, 1]$. In the case of an RBF or Sigmoid kernel, parameter $\gamma \in [0, 1]$ is optimized as well. All optimizations are conducted by means of a grid search. The detailed validation results can be found in Appendix A.3.

5.3. Test results and discussion

In Table 2 we show the classification accuracies of both reference systems, viz. k-NN(d_{BP}) and SVM($-d_{\mathrm{BP}}$), as well as the results of our novel approaches k-NN(d_{M}), SVM($-d_{\mathrm{M}}$), k-NN($s_{\varphi(g)}$), and SVM($\kappa_{\varphi(g)}$) that all rely on matching-graphs.

We observe that our novel approaches k-NN(d_{M}) and SVM($-d_{\mathrm{M}}$) outperform their respective reference systems on all data sets (except for Letter where the SVM($-d_{\mathrm{M}}$) approach achieves approximately the same accuracy as SVM($-d_{\mathrm{BP}}$)). For k-NN(d_{M}) we observe that six out of seven improvements are statistically significant, while three out of six improvements achieved with SVM($-d_{\mathrm{M}}$) are satistically significant.[6]

The classifier k-NN($s_{\varphi(g)}$) achieves higher accuracies than both reference systems on five out of seven data sets (i.e., 10 improvements in total). Five of these improvements are statistically significant. The classifier SVM($\kappa_{\varphi(g)}$) achieves even better accuracies in general. We outperform both reference systems on all data sets. Seven of the 14 improvements are statistically significant.

Comparing our novel classifiers with each other, we observe that k-NN(d_{M}) performs the best in general. That is, it outperforms both reference systems on all seven data sets, with 11 out of 14 improvements being statistically significant. Moreover, on three out of seven data sets, this classifier achieves the overall best classification results (followed by SVM($\kappa_{\varphi(g)}$) that achieves the best result in three out of seven cases, and SVM($-d_{\mathrm{M}}$) that performs best for one data set).

5.4. Ablation study, comparison with state of the art and run time analysis

We can state the following as an interim conclusion. Our novel approach using matching-graphs is clearly beneficial when com-

[6] The statistical significance is computed via Z-test using a significance level of $\alpha = 0.05$.

Table 2
Classification accuracies of two reference systems compared to our novel approaches. Symbol ○/○ indicates a statistically significant improvement and •/• indicates a statistically significant deterioration over the first and second system, respectively (using a Z-test at significance level $\alpha = 0.05$). The best result per data set is shown in bold face.

	Reference systems		Proposed system			
Data set	k-NN($\boldsymbol{d}_{\mathrm{BP}}$)	SVM($-\boldsymbol{d}_{\mathrm{BP}}$)	k-NN($\boldsymbol{d}_{\mathrm{M}}$)	SVM($-\boldsymbol{d}_{\mathrm{M}}$)	k-NN($\boldsymbol{s}_{\varphi(\mathbf{g})}$)	SVM($\boldsymbol{\kappa}_{\varphi(\mathbf{g})}$)
AIDS	98.6	99.4	**99.8** ○/−	99.7 ○/−	99.5 ○/−	99.6 ○/−
Mutagenicity	72.4	69.1	73.0 −/○	70.4 −/○	74.8 ○/○	**76.3** ○/○
NCI1	74.4	68.6	**77.6** ○/○	68.8 •/−	76.1 −/○	76.7 −/○
IMDB	60.5	63.5	68.0 ○/○	66.0 ○/−	59.0 −/−	**68.5** ○/−
COX-2	76.3	71.3	**81.7** ○/○	81.0 −/○	80.6 −/○	78.5 −/○
PTC(MR)	55.7	54.3	65.7 ○/○	**67.1** ○/○	58.6 −/−	61.4 −/−
Letter	89.9	92.7	91.7 ○/−	92.5 ○/−	90.8 ○/•	**93.2** ○/−

Table 3
Classification accuracies of an approach that uses randomly created subgraphs for embedding instead of using matching-graphs (Without-1), an approach that uses matching-graphs without using the iterative process (Without-2), as well as an approach that uses the embedded graphs without feature selection (Without-3), compared to our novel approach SVM($\kappa_{\varphi(g)}$).

Data set	Without-1	Without-2	Without-3	SVM($\boldsymbol{\kappa}_{\boldsymbol{\varphi}(\mathbf{g})}$)
AIDS	95.5 ± 0.7	99.3	99.2	**99.6**
Mutagenicity	69.1 ± 1.2	74.3	73.1	**76.3**
NCI1	70.6 ± 0.6	73.4	73.4	**76.7**
IMDB	60.9 ± 6.1	66.5	65.5	**68.5**
COX-2	73.1 ± 1.3	77.4	77.7	**78.7**
PTC(MR)	48.3 ± 1.2	44.3	44.3	**67.2**
Letter	86.1 ± 1.0	90.5	89.9	**90.1**

Table 4
Comparison of the classification accuracies of our novel framework with three state-of-the art kernel based approaches, viz. Graphlet, Shortest-Path (SP) and Wasserstein Weisfeiler–Lehman (WWL). The best accuracy per data set is shown in bold face. A dash (–) as entry indicates that the experiment timed out or that we do not get a reasonable result.

Data set	Graphlet [24]	SP [35]	WWL [36]	Ours
AIDS	98.5	99.4	99.5	**99.8**
Mutagenicity	55.5	–	**77.0**	76.3
NCI1	64.1	73.0	**79.3**	77.6
IMDB	58.0	**73.0**	71.0	68.5
COX-2	77.4	49.5	77.4	**81.7**
PTC(MR)	55.7	55.7	54.3	**67.1**
Letter	30.1	–	41.1	**93.2**

pared with similar systems that have no access to the matching-graphs. The aim of the next evaluations presented in this subsection is threefold. First, we conduct an ablation study in order to better get to the root of the strength of our novel framework. Second, we conduct a comparison with three state-of-the-art methods from the field, and third we carry out an analysis of the run time behavior.

5.4.1. Ablation study

For the embedding approach, we aim to determine whether it is the matching-graphs themselves, the iterative construction of the sets of matching-graphs or the selection of certain features that helps the most to improve the results. To this end, we conduct the following ablation study using the results of SVM($\kappa_{\varphi(g)}$) (with the subgraph based embedding).

- *Without-1*: This is a system which operates *without* matching-graphs. That is, this approach uses randomly generated subgraphs, rather than our matching-graphs, for graph embedding. The random generation of subgraphs works by randomly removing 30 to 50% of the nodes from the graphs (and their incident edges). The amount of random graphs created for each data set corresponds to the number of matching-graphs actually used for SVM($\kappa_{\varphi(g)}$). This set of random subgraphs is then used for graph embedding. We repeat the random creation of subgraphs and classification five times and report the mean and standard deviation of the accuracy.
- *Without-2*: This is a system which refrains from producing the matching-graphs with an iterative procedure as suggested in Section 3.2. Instead we use the matching-graphs created after the first for loop of Algorithm 3 (at line 6).
- *Without-3*: This is a system which operates *without* feature selection. In other words, it uses all matching-graphs created during the iterative process.

In Table 3 we see that *Without-1* performs worse compared to all other approaches on all data sets (except for the PTC(MR) data set, where the accuracy of *Without-3* is even worse). This is a first and quite strong indication of the usefulness of the matching-graphs. Next, we conclude that the iterative process on its own is not beneficial, as *Without-2* and *Without-3* perform almost equally well (except on Mutagenicity and Letter). However, feature selection applied to the resulting embedding is definitely useful as our complete system outperforms both *Without-2* and *Without-3* on all data sets (except on Letter).

In summary, the strength of our novel framework also lies in the combination of the iterative generation with a subsequent feature selection. The most valuable component of the proposed system is, however, the concept of matching-graphs themselves (as the comparisons with the reference system *Without-1* clearly show).

5.4.2. Comparison with state-of-the-art

In Table 4 we put the best accuracies of our novel framework (denoted as *Ours*) in the context with several other kernel based classifiers that are evaluated with the same experimental setup and data sets as used in the present paper. In particular, we compare our method with the *Graphlet kernel* [24], *Shortest-Path kernel (SP)* [35], as well as the *Wasserstein Weisfeiler–Lehman kernel (WWL)* [36].[7] On the Mutagenicity and NCI1 data sets our approach is narrowly outperformed by the WWL kernel and on the IMDB data set the SP kernel beats our system quite clearly. However, we can also report that on four out of seven data sets our framework achieves the overall best accuracy when compared with the current state-of-the-art.

5.4.3. Run time discussion

Of course the main downside of the proposed framework is the additional run time that comes from the increased computational demands.

[7] We compute the reference accuracies using the GraphKernels library [37] for the Graphlet and Shortest-Path kernel. For the WWL kernel we use the implementation provided in Togninalli et al. [36].

Table 5
Run time comparison of the time needed to calculate the baseline matrix $-d_{BP}$, as well as the time needed to calculate our novel matrix $-d_M$ and the times needed to create the subgraph embedding $\varphi_{sub}(g)$ and graph edit distance based embedding $\varphi_{ged}(g)$ for one graph. Time in Seconds.

	Distance based classifiers		Embedding based classifiers	
Data set	$-\boldsymbol{d}_{\mathbf{BP}}$	$-\boldsymbol{d}_{\mathbf{M}}$	$\boldsymbol{\varphi}_{\mathbf{sub}}(\mathbf{g})$	$\boldsymbol{\varphi}_{\mathbf{ged}}(\mathbf{g})$
AIDS	9	15	54	~ 0
Mutagenicity	224	240	3020	6
NCI1	552	601	633	1
IMDB	19	22	996	3
COX-2	10	15	820	3
PTC(MR)	2	3	41	~ 0
Letter	41	63	3	1

For the following discussion on the run times, we distinguish between classification systems that rely on distances and systems that are based on embeddings. For the first category we identify the following two computations whose run times are of interest.

- Run time to compute the complete distance matrix using the original graph edit distance d_{BP} by means of algorithm BP. This can be taken as a reference run time for relative comparison with the following operation.
- Run time to compute the complete distance matrix using the enhanced graph edit distance d_M using the matching-graphs.

For the second category, we are most interested in the run time for graph embedding as this is the bottleneck of our framework. In particular, we report the run times for one embedding using either the subgraph approach $\varphi_{sub}(g)$ or the graph edit distance based approach $\varphi_{ged}(g)$.

In Table 5 we show the four discussed run times of both categories in seconds. When comparing the run times for the computation of the novel distance matrix d_M with the original distance computation d_{BP}, only marginal differences can be observed. If we take into account that, for example, the k-NN that uses d_M outperforms k-NN(d_{BP}) on all seven data sets (six times with statistical significance), then this small overhead in run time is more than justified.

For the run times of the second category, weighing is more important than in the first category discussed above. For instance, it is obvious that the subgraph based embedding is significantly slower than the distance based embedding. On the other hand, we have seen that subgraph based embedding basically performs slightly better than the distance based embedding. It is not necessarily clear whether this rather small difference in accuracy can justify the large discrepancy in the run times. Note, however, that graph embedding can be parallelized with special hardware infrastructure, which in turn can dramatically reduce the high run time, if necessary.

6. Conclusion and future work

In the present paper we introduce and research a novel data structure called matching-graph, which can be pre-computed on training graphs. Our general goal is to leverage the power of graph edit distance to build a novel graph representation that formalizes the matching parts found between two graphs. This formalization can be interpreted as stable part, or core, of two graphs. We propose to build matching-graphs on the basis of the edit path between two graphs. Formally, a matching-graph of two graphs consists of the nodes substituted under a given cost model in a graph edit distance computation. We propose to build matching-graphs between all pairs of graphs stemming from the same class. Eventually, we define two complementary approaches that (a) condense the initial set of matching-graphs to the most influential ones and (b) iteratively enlarges the set of matching-graphs by recursively building novel matching-graphs out of already created matching-graphs. The benefit of these matching-graphs is that they can be utilized to improve the classification accuracy of various classifiers. The drawback is – of course – the increased computation time.

To show the usefulness of matching-graphs we propose two classification approaches. The first system employs a weighted distance of the original graph edit distance and an aggregated distance to sets of matching-graphs. The second approach uses the matching-graphs to build vector representations of the underlying graphs. To this end, we embed our graphs in an N-dimensional vector space such that the i-th entry of the resulting vector represents either the distance to the corresponding matching-graph or whether the corresponding matching-graph occurs as a subgraph in the graph to be embedded.

By means of a thorough experimental evaluation on diverse graph data sets covering a wide spectrum of applications, we empirically confirm that classification systems that (in part) rely on the novel matching-graphs significantly outperform their counterparts that have no access to this specific information.

In a thorough ablation study, we are also able to clearly underline the value of the novel matching-graphs. Last but not least, with a comparison with three state-of-the-art methods we empirically confirm that our framework is able to set new benchmarks on several data sets.

The proposed matching-graphs have – besides the ability of improving the classification accuracy in a graph-based classification scenario – another interesting benefits. They can automatically reveal significant patterns in large sets of graphs. In particular, it turns out that matching-graphs often represent crucial patterns that actually constitute a certain class of patterns. For instance, for the mutagen class of the Mutagenicity data set we autonomically identified both patterns NO_2 and NH_2 in many matching-graphs. Both compounds are well known to be mutagenic [38]. This is especially interesting as the matching-graphs are automatically created on the basis of the edit path between training graphs without any domain knowledge. In future work we plan to exploit this automatic finding of significant patterns in the data in a more systematic manner (and discuss the identified patterns with domain experts). Besides this idea we identify the following potential future research activities.

- Rather than the approximation algorithm BP, one could employ any other graph edit distance computation. We believe that using more expensive algorithms for the computation of the edit distance could lead to other (perhaps larger?) matching-graphs.
- The novel matching-graphs might actually be used as a sparse representation of any input data. Hence, our framework could potentially be used as a novel and quite fast way for dictionary learning in the graph domain.

- We plan to conduct more theoretical and empirical investigation of the novel matching-graphs to further explore the relationship between matching-graphs and common subgraphs.
- There are several other ways to use the matching-graphs for classification purposes. For instance, we could employ the matching-graphs to augment training sets for classifiers that are heavily dependent on large sets of training data (e.g. graph neural networks).
- Currently, we aim at finding a data structure that represents the similarity core of two given graphs and thus we only consider nodes that are actually matched with each other under some cost model. We do see the potential of building matching-graphs that do not only include substituted nodes, but also inserted (or deleted) nodes.
- Last but not least, we feel, that the concept of matching-graphs might also be beneficial for regression problems (e.g. one could employ the matching-graphs in conjunction with a nearest-neighbor regression, which depends on a large number of training data).

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Appendix A. Detailed validation results

A1. Validation results k-NN($d_{\mathbf{M}}$) and SVM($-d_{\mathbf{M}}$)

In Tables A.6 and A.7 we show the best performing parameters found during validation for k-NN(d_{M}) and SVM($-d_{\mathrm{M}}$), respectively. Major findings of the validation process for the k-NN are as follows(see also Table A.6). On all data sets but NCI1 the parameter α lies between 0.35 and 0.65, which means that both distance informations are – more or less – equally important. Moreover, we observe that pruning seems to be beneficial in general, and that the optimal selection method is center on all data sets but NCI1. Finally we can see that the max function performs the best for distance aggregation in almost all cases.

The optimal parameters for SVM (see Table A.7) indicate that the aggregated distance is rather less important and that the optimal selection method is center in most of the cases. Furthermore, the min, together with the max function, is often used for condensing the set of distances.

Table A3
The number of matching-graphs created for each data set and the final number of matching-graphs after feature selection is applied.

Data set	Total	Selected
AIDS	4,955	199
Mutagenicity	86,752	4139
NCI1	4544	618
IMDB	43,015	2141
COX-2	19,704	989
PTC(MR)	3893	207
Letter	13,514	689

A2. Number of graphs after feature selection

In Table A.8 we show the total number of matching-graphs produced first and the number of matching-graphs selected (via recursive feature elimination). On all data sets substantial reductions can be observed. For instance, on AIDS, Mutagenicity, IMDB, COX-2, PTC(MR) and Letter about 4 to 5% of the available matching-graphs are selected, while on NCI1 about 13% of the matching-graphs are selected. The conclusions we draw from this table are twofold. First, we note that the iterative procedure can be used to produce almost arbitrarily large sets of graphs. Second, it appears that only a small fraction of the matching-graphs are actually needed. Of course, it would be desirable to produce from the very start only those matching-graphs that will really be used' with our current solution this is not possible and we thus follow the well-known paradigm of *overproduce and select*.

A3. Validation results k-NN($s_{\varphi(\mathbf{g})}$) and SVM($\kappa_{\varphi(\mathbf{g})}$)

In Tables A.9 and A.10 we show the optimal parameters for k-NN($s_{\varphi(g)}$) and SVM($\kappa_{\varphi(g)}$). For the k-NN($s_{\varphi(g)}$) (see Table A.9) the embedding applied is $\varphi_{sub}(g)$ for all data sets except for Letter. Regarding the similarity measure s we can not report a clear winner (although Kulczynski-1 or -2 might be a good choice when in doubt).

Table A1
Optimal parameter values found on the validation sets for the k-NN(d_{M}) classifier (including the classification accuracy).

Data set	β	α	Pruning	Selection	t	S	k	Accuracy
AIDS	0.90	0.35	pruned	center	30	max	7	99.6
Mutagenicity	0.60	0.60	pruned	center	10	max	1	76.2
NCI1	0.75	0.90	pruned	spanning	3	avg	1	78.5
IMDB	0.40	0.65	unpruned	center	65	max	3	75.5
COX-2	0.40	0.50	pruned	center	10	avg	5	84.9
PTC(MR)	0.60	0.30	pruned	center	7	max	7	69.1
Letter	0.65	0.60	pruned	center	60	max	3	93.2

Table A2
Optimal parameter values found on the validation sets for the SVM($-d_{\mathrm{M}}$) classifier (including the classification accuracy).

Data set	β	α	Pruning	Selection	t	S	C	Accuracy
AIDS	0.60	0.85	pruned	spanning	75	min	10^{-1}	100.0
Mutagenicity	0.75	0.65	pruned	center	15	max	10^{-2}	73.2
NCI1	0.75	0.95	pruned	center	75	max	10^{-2}	67.3
IMDB	0.35	0.90	pruned	center	65	avg	10^{-3}	70.0
COX-2	0.95	0.90	pruned	center	25	max	10^{-1}	82.8
PTC(MR)	0.75	0.90	unpruned	spanning	25	min	10^{1}	77.9
Letter	0.70	0.90	pruned	spanning	75	min	10^{-2}	94.3

Table A4
Optimal parameter values found on the validation sets for the k-NN($s_{\varphi(g)}$) classifier (including the classification accuracy).

Data set	Embedding	β	s	k	Accuracy
AIDS	φ_{sub}	N/A	Kulczynski-1	5	99.6
Mutagenicity	φ_{sub}	N/A	Dice	7	79.2
NCI1	φ_{sub}	N/A	Rogers	5	79.2
IMDB	φ_{sub}	N/A	Kulczynski-2	7	76.0
COX-2	φ_{sub}	N/A	Yule	5	82.8
PTC(MR)	φ_{sub}	N/A	Kulczynski-2	5	72.1
Letter	φ_{ged}	0.70	Cosine	5	91.2

Table A5
Optimal parameter values found on the validation sets for the SVM($\kappa_{\varphi(g)}$) classifier (including the classification accuracy).

Data set	Embedding	β	Kernel	C	γ	Accuracy
AIDS	φ_{sub}	N/A	RBF	10^{-1}	10^{0}	99.6
Mutagenicity	φ_{sub}	N/A	RBF	10^{2}	10^{-3}	82.4
NCI1	φ_{sub}	N/A	RBF	10^{1}	10^{-2}	77.4
IMDB	φ_{sub}	N/A	RBF	10^{1}	10^{-3}	74.5
COX-2	φ_{ged}	0.40	Linear	10^{-2}	N/A	86.0
PTC(MR)	φ_{ged}	0.05	Linear	5×10^{-3}	N/A	77.8
Letter	φ_{ged}	0.60	Linear	5×10^{-3}	N/A	93.1

The optimal parameters for the SVM (see Table A.10) indicate that the best embedding function is on four out of seven data sets the subgraph based function φ_{sub}. The best performing kernel function κ is in four out of seven data sets the RBF kernel, which are notably all based on the φ_{sub} embedding. On the three other data sets that use φ_{ged} for embedding, the linear kernel seems to be optimal. The best value of C is either 10 or 100 for all data sets embedded with the φ_{sub} embedding, except for AIDS, where the optimal value is 0.1. For the φ_{ged} embedded graphs the optimal values for C are much smaller (between 0.005 and 0.01). The optimal parameter γ on the other hand is smaller than 0.01 for all data sets except for AIDS (where $\gamma = 1$ performs the best).

References

[1] S. Xie, H. Hu, Y. Wu, Deep multi-path convolutional neural network joint with salient region attention for facial expression recognition, Pattern Recognit. 92 (2019) 177–191, doi:10.1016/j.patcog.2019.03.019.

[2] K.G. Lore, A. Akintayo, S. Sarkar, LLNet: a deep autoencoder approach to natural low-light image enhancement, Pattern Recognit. 61 (2017) 650–662, doi:10.1016/j.patcog.2016.06.008.

[3] C. Li, J. Guo, F. Porikli, Y. Pang, LightenNet: a convolutional neural network for weakly illuminated image enhancement, Pattern Recognit. Lett. 104 (2018) 15–22, doi:10.1016/j.patrec.2018.01.010.

[4] N. Shervashidze, P. Schweitzer, E.J. van Leeuwen, K. Mehlhorn, K.M. Borgwardt, Weisfeiler–Lehman graph kernels, J. Mach. Learn. Res. 12 (2011) 2539–2561.

[5] P. Maergner, V. Pondenkandath, M. Alberti, M. Liwicki, K. Riesen, R. Ingold, A. Fischer, Combining graph edit distance and triplet networks for offline signature verification, Pattern Recognit. Lett. 125 (2019) 527–533, doi:10.1016/j.patrec.2019.06.024.

[6] M. Curado, F. Escolano, M.A. Lozano, E.R. Hancock, Early detection of Alzheimer's disease: detecting asymmetries with a return random walk link predictor, Entropy 22 (4) (2020) 465, doi:10.3390/e22040465.

[7] D. Conte, P. Foggia, C. Sansone, M. Vento, Thirty years of graph matching in pattern recognition, Int. J. Pattern Recognit. Artif. Intell. 18 (3) (2004) 265–298, doi:10.1142/S0218001404003228.

[8] P. Foggia, G. Percannella, M. Vento, Graph matching and learning in pattern recognition in the last 10 years, Int. J. Pattern Recognit. Artif. Intell. 28 (1) (2014), doi:10.1142/S0218001414500013.

[9] H. Bunke, G. Allermann, Inexact graph matching for structural pattern recognition, Pattern Recognit. Lett. 1 (4) (1983) 245–253, doi:10.1016/0167-8655(83)90033-8.

[10] T. Gärtner, Kernels for structured data, Series in Machine Perception and Artificial Intelligence, vol. 72, WorldScientific, 2008, doi:10.1142/6855.

[11] F. Scarselli, M. Gori, A.C. Tsoi, M. Hagenbuchner, G. Monfardini, The graph neural network model, IEEE Trans. Neural Networks 20 (1) (2009) 61–80, doi:10.1109/TNN.2008.2005605.

[12] M. Fuchs, K. Riesen, Matching of matching-graphs—A novel approach for graph classification, in: 25th International Conference on Pattern Recognition, ICPR 2020, Virtual Event / Milan, Italy, January 10–15, 2021, IEEE, 2020, pp. 6570–6576, doi:10.1109/ICPR48806.2021.9411926.

[13] M. Fuchs, K. Riesen, Iterative creation of matching-graphs – finding relevant substructures in graph sets, in: In Proceedings of the 25th Iberoamerican Congress on Pattern Recognition, CIARP25 2021, 2021.

[14] M. Fuchs, K. Riesen, et al., Graph embedding in vector spaces using matching-graphs, in: N. Reyes (Ed.), Similarity Search and Applications - 14th International Conference, SISAP 2021, Dortmund, Germany, September 29–October 1, 2021, Proceedings, LNCS, vol. 13058, Springer, 2021, pp. 352–363, doi:10.1007/978-3-030-89657-7_26.

[15] K. Riesen, H. Bunke, Approximate graph edit distance computation by means of bipartite graph matching, Image Vis. Comput. 27 (7) (2009) 950–959, doi:10.1016/j.imavis.2008.04.004.

[16] A. Fischer, C.Y. Suen, V. Frinken, K. Riesen, H. Bunke, Approximation of graph edit distance based on Hausdorff matching, Pattern Recognit. 48 (2) (2015) 331–343, doi:10.1016/j.patcog.2014.07.015.

[17] S. Bougleux, B. Gaüzère, D.B. Blumenthal, L. Brun, Fast linear sum assignment with error-correction and no cost constraints, Pattern Recognit. Lett. 134 (2020) 37–45, doi:10.1016/j.patrec.2018.03.032.

[18] U. Kang, M. Hebert, S. Park, Fast and scalable approximate spectral graph matching for correspondence problems, Inf. Sci. 220 (2013) 306–318, doi:10.1016/j.ins.2012.07.008.

[19] A. Solé-Ribalta, F. Serratosa, Graduated assignment algorithm for multiple graph matching based on a common labeling, Int. J. Pattern Recognit. Artif. Intell. 27 (1) (2013), doi:10.1142/S0218001413500018.

[20] B. Luo, E.R. Hancock, Structural graph matching using the EM algorithm and singular value decomposition, IEEE Trans. Pattern Anal. Mach. Intell. 23 (10) (2001) 1120–1136, doi:10.1109/34.954602.

[21] N.M. Kriege, C. Morris, et al., Recent advances in kernel-based graph classification, in: Y. Altun (Ed.), Machine Learning and Knowledge Discovery in Databases - European Conference, ECML PKDD 2017, Skopje, Macedonia, September 18–22, 2017, Proceedings, Part III, LNCS, vol. 10536, Springer, 2017, pp. 388–392, doi:10.1007/978-3-319-71273-4_37.

[22] M. Deshpande, M. Kuramochi, N. Wale, G. Karypis, Frequent substructure-based approaches for classifying chemical compounds, IEEE Trans. Knowl. Data Eng. 17 (8) (2005) 1036–1050, doi:10.1109/TKDE.2005.127.

[23] N.M. Kriege, P. Mutzel, Subgraph matching kernels for attributed graphs, in: Proceedings of the 29th International Conference on Machine Learning, ICML 2012, Edinburgh, Scotland, UK, June 26–July 1, 2012, icml.cc / Omnipress, 2012.

[24] N. Shervashidze, S.V.N. Vishwanathan, T. Petri, K. Mehlhorn, K.M. Borgwardt, Efficient graphlet kernels for large graph comparison, in: D.A.V. Dyk, M. Welling (Eds.), Proceedings of the Twelfth International Conference on Artificial Intelligence and Statistics, 2009, Florida, USA, April 16–18, 2009, JMLR Proceedings, vol. 5, JMLR.org, 2009, pp. 488–495.

[25] K. Riesen, H. Bunke, Classification and clustering of vector space embedded graphs, in: Emerging Topics in Computer Vision and its Applications, World Scientific, 2012, pp. 49–70, doi:10.1142/9789814343008_0003.

[26] M. Moussaoui, M. Zaghdoud, J. Akaichi, A survey of uncertainty handling in frequent subgraph mining algorithms, in: 12th IEEE/ACS International Conference of Computer Systems and Applications, AICCSA 2015, Marrakech, Morocco, November 17–20, 2015, IEEE Computer Society, 2015, pp. 1–8, doi:10.1109/AICCSA.2015.7507186.

[27] A. Inokuchi, T. Washio, H. Motoda, An apriori-based algorithm for mining frequent substructures from graph data, in: D.A. Zighed, H.J. Komorowski, J.M. Zytkow (Eds.), Principles of Data Mining and Knowledge Discovery, 4th European Conference, PKDD 2000, Lyon, France, September 13–16, 2000, Proceedings, LNCS, vol.1910, Springer, 2000, pp. 13–23, doi:10.1007/3-540-45372-5_2.

[28] D.J. Cook, L.B. Holder, Substructure discovery using minimum description length and background knowledge, J. Artif. Intell. Res. 1 (1994) 231–255, doi:10.1613/jair.43.

[29] H. Bunke, On a relation between graph edit distance and maximum common subgraph, Pattern Recognit. Lett. 18 (8) (1997) 689–694, doi:10.1016/S0167-8655(97)00060-3.

[30] K. Riesen, H. Bunke, et al., IAM graph database repository for graph based pattern recognition and machine learning, in: N. da Vitoria Lobo (Ed.), Structural, Syntactic, and Statistical Pattern Recognition, Joint IAPR International Workshop, SSPR & SPR 2008, Orlando, USA, December 4–6, 2008. Proceedings, LNCS, vol.5342, Springer, 2008, pp. 287–297, doi:10.1007/978-3-540-89689-0_33.

[31] X. Jiang, A. Münger, H. Bunke, On median graphs: properties, algorithms, and applications, IEEE Trans. Pattern Anal. Mach. Intell. 23 (10) (2001) 1144–1151, doi:10.1109/34.954604.

[32] L.P. Cordella, P. Foggia, C. Sansone, M. Vento, A (sub)graph isomorphism algorithm for matching large graphs, IEEE Trans. Pattern Anal. Mach. Intell. 26 (10) (2004) 1367–1372, doi:10.1109/TPAMI.2004.75.

[33] C. Morris, N.M. Kriege, F. Bause, K. Kersting, P. Mutzel, M. Neumann, Tu-dataset: a collection of benchmark datasets for learning with graphs, ICML 2020 Workshop on Graph Representation Learning and Beyond (GRL+ 2020), 2020. http://www.graphlearning.io

[34] R.Y. Krisnabayu, A. Ridok, A.S. Budi, Hepatitis detection using random forest based on SVM-RFE (recursive feature elimination) feature selection and SMOTE, in: SIET '21: 6th International Conference on Sustainable Information Engineering and Technology 2021, Malang, Indonesia, September 13–14, 2021, ACM, 2021, pp. 151–156, doi:10.1145/3479645.3479668.

[35] K.M. Borgwardt, H.-P. Kriegel, Shortest-path kernels on graphs, in: Proceedings of the 5th IEEE International Conference on Data Mining (ICDM 2005), 27–30 November 2005, Houston, Texas, USA, IEEE Computer Society, 2005, pp. 74–81, doi:10.1109/ICDM.2005.132.

[36] M. Togninalli, E. Ghisu, F. Llinares-López, B. Rieck, K. Borgwardt, Wasserstein Weisfeiler–Lehman graph kernels, in: H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché Buc, E. Fox, R. Garnett (Eds.), Advances in Neural Information Processing Systems 32 (NeurIPS), Curran Associates, Inc., 2019, pp. 6436–6446.
[37] M. Sugiyama, M.E. Ghisu, F. Llinares-López, K.M. Borgwardt, Graphkernels: R and python packages for graph comparison, Bioinform. 34 (3) (2018) 530–532, doi:10.1093/bioinformatics/btx602.
[38] D. Luo, W. Cheng, D. Xu, W. Yu, B. Zong, H. Chen, X. Zhang, Parameterized explainer for graph neural network, in: H. Larochelle, M. Ranzato, R. Hadsell, M.-F. Balcan, H.-T. Lin (Eds.), Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6–12, 2020, Virtual, 2020.

Kaspar Riesen received his M.Sc. and Ph.D. degrees in Computer Science from the University of Bern in 2006 and 2009, respectively. His Ph.D. thesis received the Alumni price for an outstanding work at the Institute of Computer Science and Applied Mathematics of the University of Bern. His research interests cover the fields of artificial intelligence, pattern recognition, machine learning and data mining. In particular, he is working on the development of novel algorithms for solving graph matching problems in various domains of intelligent information processing.

Mathias Fuchs has received a B.Sc. and M.Sc. in Computer Science from the University of Bern in 2017 and 2019, respectively. He is currently a Ph.D. student in the Pattern Recognition Group at the University of Bern. His research interests are pattern recognition with a special focus on graph matching and graph representations.